Escuela
Politécnica
Superior

Registro de datos 3D para la visualización de la escena de un crimen

Fin de Grado

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Jose Manuel Puentes Pérez

Tutor/es:
Miguel Angel Cazorla Quevedo

Universitat d'Alacant
Universidad de Alicante

Enero 2018

Registro de datos 3D
Para la visualización de la escena de un crimen.

UNIVERSIDAD DE ALICANTE
CURSO 2017-2018

DCCIA

Tutor
Miguel Ángel Cazorla Quevedo

Alumno
Jose Manuel Puentes Pérez

Resumen

Se trata de un proyecto en el que se quiere crear una aplicación para la toma de datos en formato pcd “Point Cloud” con sensor Kinect V1 y ser usados para su reconstrucción y la visualización en 3D con el fin de guardar la información de un escenario de un crimen. Estas reconstrucciones usan la librería PCL y su facilidad de manejo con datos en formato 3D. La aplicación estará dispuesta de tal forma que se pueda configurar según los datos que hayamos tomado o si ya disponemos de datos. Se deberá usar metodos lo suficientemente flexible para que se pueda usar en distintos escenarios. Intentaremos reconstruir varios escenarios para probar dicha aplicación y veremos si somos capaces de hacerlo con los pocos recursos a nuestra disposición.

Palabras clave

Registro 3D, visión artificial, PCL, Kinect, Reconstrucción.

Índice

1. Introducción

1.1. Motivación

Se pretende desarrollar una aplicación que sea capaz de procesar nubes de puntos tomadas con un sensor tipo kinect para completar una herramienta ya desarrollada capaz de, a partir de datos aportados por biólogos forenses, determinar el intervalo post mortem de un cadáver a partir de los insectos encontrados en su cuerpo. Esta nueva aplicación capturará y visualizará en 3D el cadáver de la escena de un crimen. Con ello se pretende tener más información sobre dicho escenario.

Para ello, usaremos la librería PCL (procesamiento de datos 3D) que proporciona las herramientas básicas de manejo de datos y el sensor Kinect como principal sensor.

Esta motivación se debe a que durante la asignatura VAR (Visión Artificial y Robótica) se realizó una práctica en la que se usaba ROS, turtlebot (Robot virtual con Kinect) y la PCL para crear una panorámica y para intentar reconstruir una habitación virtual mediante nubes de puntos. Esto me atrayó y me planteé hacer un TFG que estuviese relacionado con la Kinect y la PCL.

1.2. Objetivos generales

Se ha desarrollado una aplicación que se encarga de la toma de datos de un escenario junto a una Kinect como principal sensor, para la reconstrucción del escenario en el que se encuentra un cadáver. Se necesita un equipo portátil para acceder al escenario. Una vez hemos tomado estos datos con la aplicación la configuramos según las características de dichos datos para su reconstrucción y visualización en un modelo 3D del cadáver de la escena, se puede hacer en el mismo portátil o en un ordenador más potente, según la calidad que necesitemos.

Para lograr el objetivo, se va a diseñar e implementar una aplicación que en modo captura de datos, permite al usuario conectar la Kinect al portátil y visualizar los datos a través de la herramienta, tomando datos del objeto a reconstruir de una manera determinada. Una vez tomado estos datos se ejecuta esta misma aplicación en su modo reconstrucción que accede a dichos datos. A continuación se configura según la calidad de los datos y el tamaño de los mismos, y se comprueba que funciona correctamente, una vez comprobado que funciona se ejecuta hasta su total reconstrucción.

1.3. Metodología

Respecto a los pasos que se han seguido para la realización de este trabajo:

En primer lugar, se ha realizado un estudio sobre la Kinect al querer capturar la información con dicho sensor, también se ha tenido que explorar los pormenores del dispositivo y analizar si efectivamente sirve a nuestros propósitos. A continuación, se crea la primera parte de la aplicación que consiste en capturar las imágenes, se han capturado 3 grupos de datos; treinta y seis imágenes de una persona sentada, setenta y tres imágenes de un camión de juguete a dos alturas distintas intercaladas, y setenta y cuatro imágenes de un muñeco, primero una vuelta a una altura y la siguiente vuelta a otra altura. Los datos de los dos juguetes han sido tomados apoyando la Kinect en un trípode y aproximadamente cada 10 grados, las de la persona han sido tomados al aire.

Estas capturas constituirán nuestros escenarios con los objetos a reconstruir, en tres tamaños distintos ordenados de mayor a menor; persona, camion, muñeco. Una vez tenemos los datos, lo siguiente es ver qué métodos de reconstrucción se pueden emplear, realizar unas cuantas reconstrucciones y ver qué método se ajusta mejor a nuestros requerimientos. Una vez elegido el método probaremos configuraciones a ver hasta dónde puede llegar la reconstrucción.

Figura 1. *Ejemplo crimen.*
http://tidop.usal.es/reconstruct-the-crime-scene-in-three-dimensions

1.3.1. KINECT

Consta de una cámara RGB y un sensor de profundidad, que consiste en un proyector de láser de infrarrojos combinado con un sensor CMOS que captura la información en 3D bajo cualquier condición de iluminación.

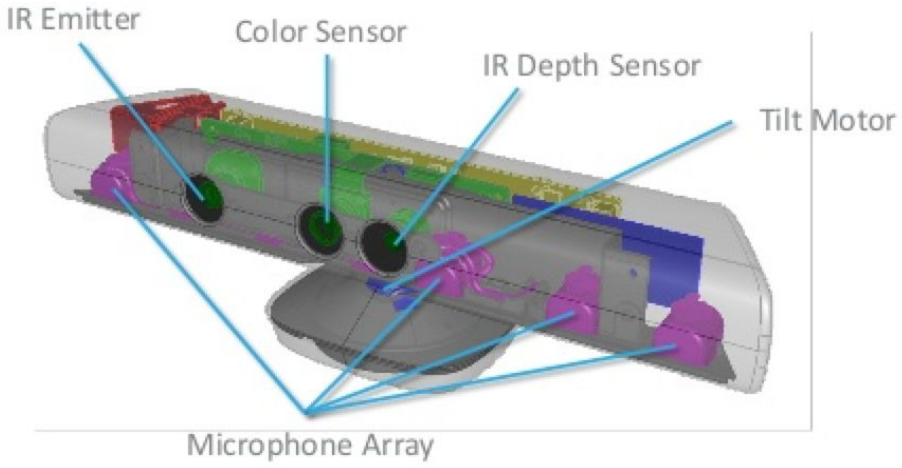


Figura 2. *Cámara Kinect*

Proporciona un campo de visión horizontal de 57 grados, un campo de visión vertical de 43 grados y un rango de profundidad entre 1.2 metros y 3.5 metros. La resolución de las imágenes de color es de 640 x 480 píxeles. A la hora de la captura de nubes de puntos el dispositivo devuelve un máximo de 307.200 puntos.

Este dispositivo nos sirve para la captura de imágenes RGB y de profundidad, con el fin de reconocer objetos o figuras para la realización de una reconstrucción. Se conecta al ordenador mediante cable USB, y puede colocarse donde se desee pero lo ideal sería en un trípode para tener siempre la misma altura y facilitar el trabajo.

1.3.2. PCL

PCL (Point Cloud Library) es una librería para el procesamiento de imágenes y nubes de puntos ya sea en 2D o 3D. PCL contiene numerosos algoritmos para el tratamiento de estas nubes como filtros, extracción de características, reconstrucción de superficies, registro, ajuste de modelos, segmentación etc.

Esta librería hay que instalarla, desde el apt-get, o descargando, compilando y ejecutando la versión que necesitemos. En nuestro caso hemos usado la version PCL 1.7.

El uso que le vamos a dar será el tratamiento de las imágenes de la Kinect para aplicarles transformaciones e ir sumando entre ellas, para recrear el escenario u objeto.

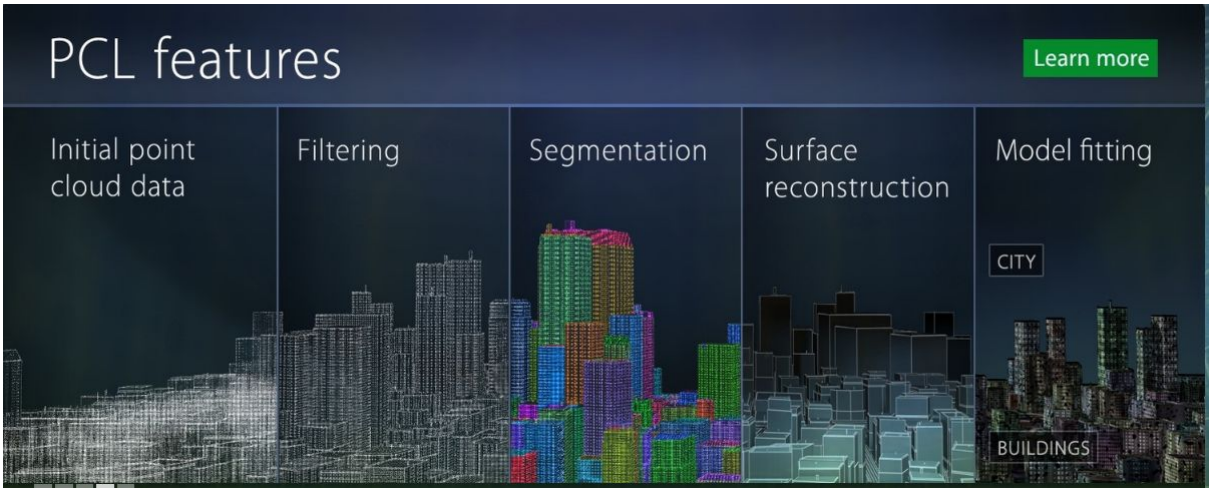


Figura 3. *PCL Features*

1.4. Datos usados

Para la experimentación con la aplicación necesitamos datos que contengan escenario y un modelo para reproducción de las condiciones del entorno. Esta aplicación trata de reconstruir a la persona. Para la toma de datos se han usado varias formas de hacerlo:

- Toma de datos con un trípode para tener constante la altura y el ángulo.
- Toma de datos con trípode con dos alturas.
- Toma de datos sin trípode.

¿Por qué hemos usado estos tres métodos?

Esto es debido a que mediante la sujeción con un trípode tenemos dos factores fijos cuando tomamos fotografías del escenario, esto quiere decir que la diferencia entre las dos nubes de puntos que vamos a tratar de unir será menor ya que la altura y el ángulo con respecto al suelo siempre será el mismo o muy parecido. Serían un conjunto de control.

El tercer conjunto es el más realista, puesto que la toma de datos se hace al aire sujetando la Kinect con las manos, por lo que puede variar la distancia al objeto, la distancia al suelo, el ángulo de la foto hacia el objeto …

Todo esto nos va a permitir estudiar el error cometido con cada método.

Estas tomas de datos han sido con el objeto en el centro de una habitación, sobre una tela con marcas radiales cada diez grados, por lo que tenemos controlados varios factores. En cuanto a los objetos que se han usado son:

- Una persona:
  Tamaño aproximado:
  - Grande
  - Altura ≃ 80cm
  - Ancho ≃ 65cm
  - Profundo ≃ 57cm
  - Número de imágenes tomadas 36.

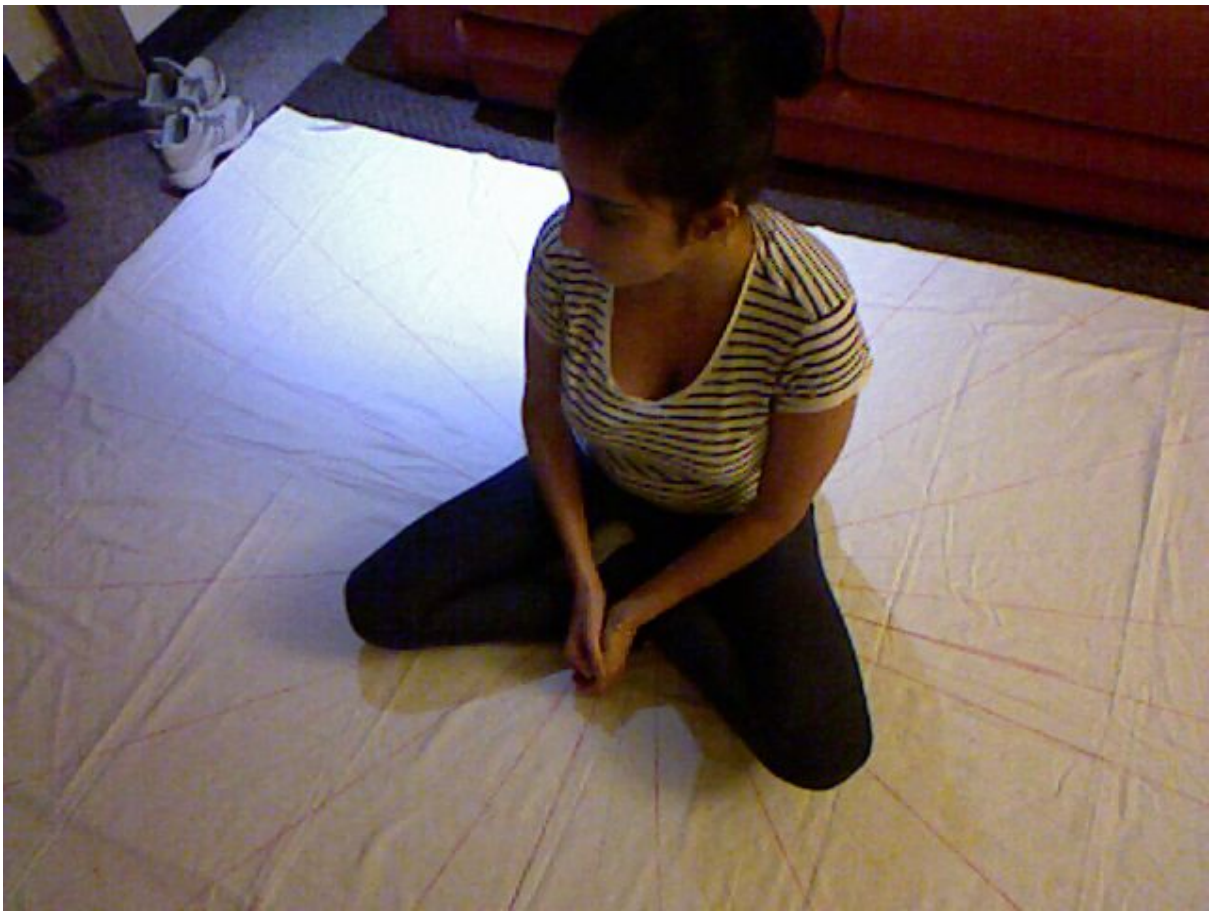


Figura 4. *Persona, Captura con nombre snap_point2.pcd*

- Un camión de juguete:
  Tamaño aproximado:
  - Mediano
  - Altura ≃ 40cm
  - Ancho ≃ 33cm
  - Profundo ≃ 90cm
  - Número de imágenes tomadas 73.

Figura 5. *Camión, Captura con nombre snap_point8.pcd*

- Un juguete de Mikey:
  Tamaño aproximado:
  - Pequeño
  - Altura ≃ 26cm
  - Ancho ≃ 15cm
  - Profundo ≃ 12cm
  - Número de imágenes tomadas 74.

Figura 6. *Mikey, Captura con nombre snap_point4.pcd*

Las nubes de puntos que proceden de la Kinect v1 contienen un máximo de 307200 puntos, un ancho de 640 y alto 480 pixeles. Estos puntos se componen de componente x, y, z y rgba, esto los posiciona en el espacio y les da un color.

Debido a que la Kinect es la V1 esta tiene una resolución muy pobre y la que se ha usado para la toma de datos tiene fallos, por lo que no tenemos todos los puntos que deberíamos obtener. También hay que añadir que se necesita una distancia mínima desde el sensor hacia su entorno, por lo que si hay objetos muy cerca o muy lejos del mismo estos puntos serán del tipo nan (Not a Number), punto no válido. Seguidamente veremos una visualización de dichas nubes de puntos de los tres objetos:

- Persona.

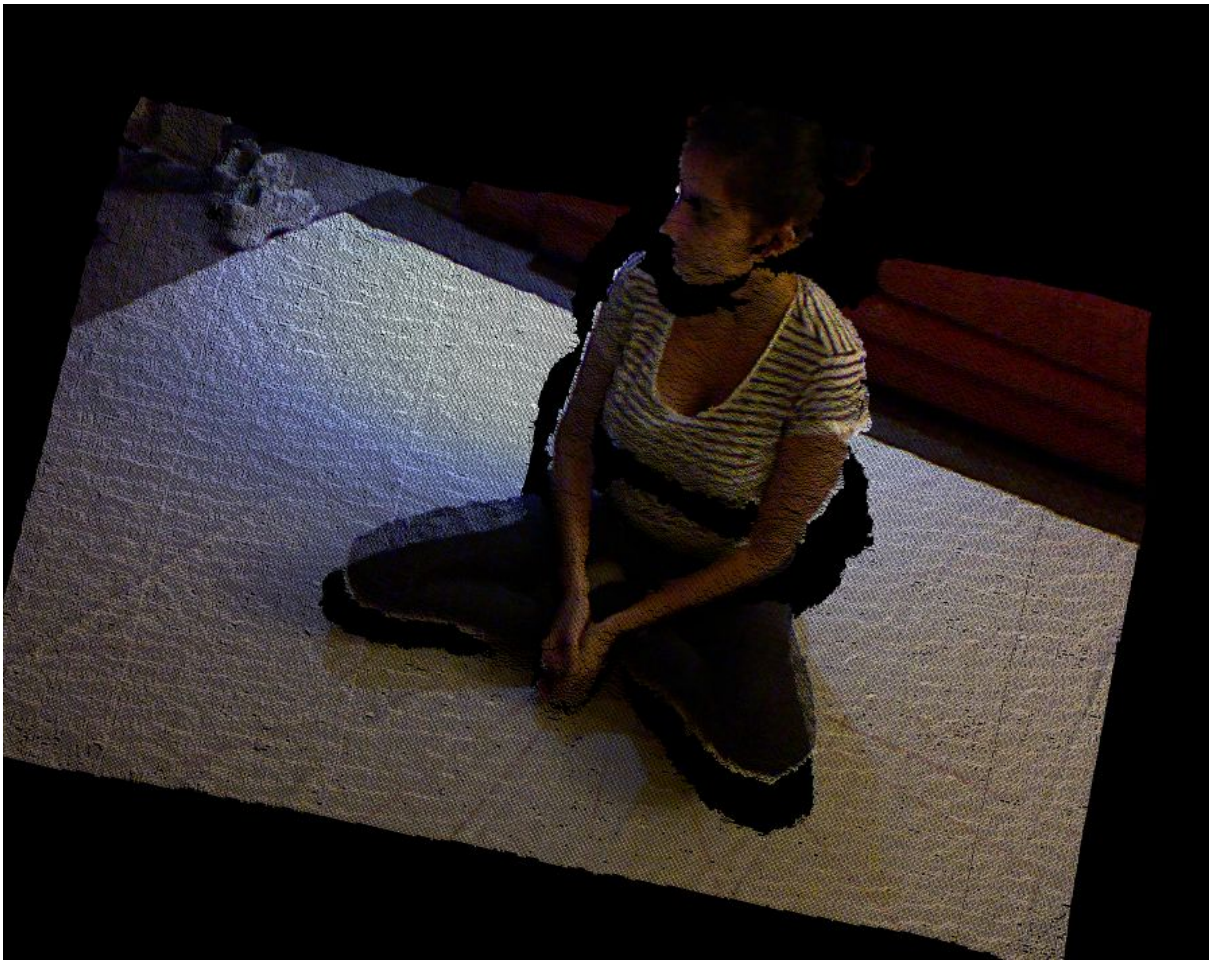


Figura 7. *Persona, Captura con nombre snap_point2.pcd*

Como se aprecia en la imágen hay huecos en negro, esos son los puntos que no se pueden alcanzar con el ángulo actual del sensor.

- Camión de juguete.

Figura 8. *Camión, Captura con nombre snap_point8.pcd*

Como vemos en esta imágen, los puntos de la cabeza del camión no se ven, ya que no han sido recogidos debido a las limitaciones del sensor y de los errores del mismo.

- Mikey de juguete.

Figura 9. *Mikey, Captura con nombre snap_point4.pcd*

En esta otra imágen vemos sin embargo otro tipo de error, que son los conjuntos de datos que se toman erróneamente, como los conjuntos que aparecen más abajo del suelo. Estos puntos son distancias que no están bien calculadas ya sea por el material que hay en el suelo o por el error del sensor. Esto hace que sea más difícil de tratar las imágenes.

2. Estructura

2.1. Captura de datos y visualización de la Kinect.

Se ha usado OpenNI Grabber Framework en la PCL para la conexión de la Kinect al código y así poder tomar datos de dicho sensor.

A partir de PCL 1.0, ofrecen una nueva interfaz de captura genérica para proporcionar un acceso fácil y conveniente a diferentes dispositivos y sus controladores, sus formatos de archivo y otras fuentes de datos.

El primer controlador que incorpora es el nuevo OpenNI Grabber, que hace que sea muy fácil solicitar transmisiones de datos desde cámaras compatibles con OpenNI. En este proyecto se configura y usa el capturador, y como es tan simple, se logra hacer en pocas líneas de código. Estos métodos aceptan sensores como “Primesense Reference Design”, “Microsoft Kinect” y “Asus Xtion Pro”.

Lo siguiente es guardar los datos que estamos capturando en archivos en forma de nubes de puntos.

2.2. Tratamiento de datos y reconstrucción de los mismos.

Para el tratamiento de estas nubes de puntos tenemos que conocer los siguientes conceptos:

2.2.1. Point cloud (Nubes de puntos)

¿Qué es una nube de puntos?

Una nube de puntos es una estructura de datos utilizada para representar una colección de puntos multidimensionales y se usa comúnmente para representar datos tridimensionales. En una nube de puntos 3D, los puntos generalmente representan las coordenadas geométricas X, Y y Z de una superficie muestreada subyacente. Cuando la información de color está presente (ver las figuras a continuación), la nube de puntos se convierte en puntos RGB.

Figura 10, *Ejemplo PointCloud.* Esta nube se guarda en archivos de extensión “pcd”.

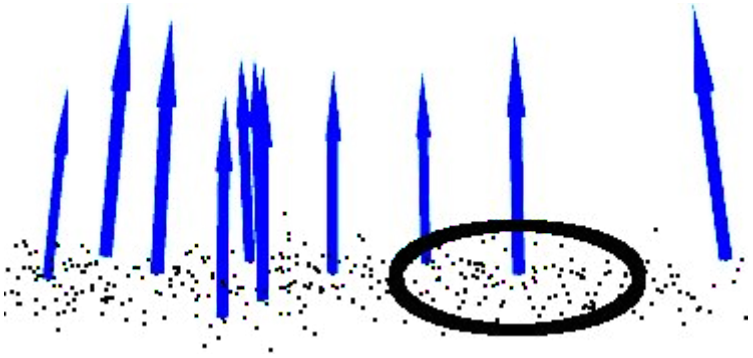
2.2.2. Keypoints (Puntos Clave)

La biblioteca pcl_keypoints contiene implementaciones de dos algoritmos de detección de punto clave de nube de puntos. Los puntos clave (también denominados puntos de interés) son puntos en una imagen o nube de puntos que son estables, distintivos y pueden identificarse utilizando un criterio de detección bien definido. Normalmente, el número de puntos de interés en una nube de puntos será mucho más pequeño que el número total de puntos en la nube, y cuando se usan en combinación con descriptores de características locales en cada punto clave, los puntos clave y descriptores se pueden usar para formar una nube compacta de los datos originales.

2.2.3. Features (Descriptores)

La biblioteca pcl_features contiene estructuras de datos y mecanismos para la estimación de características 3D a partir de datos de nubes de puntos. Las características 3D son representaciones en un cierto punto 3D o posición en el espacio, que describen patrones geométricos basados en la información disponible alrededor del punto. El espacio de datos seleccionado alrededor del punto de consulta generalmente se conoce como el k-vecindad.

La siguiente figura muestra un ejemplo simple de un punto de consulta seleccionado, y su k-vecindad seleccionado.



Un ejemplo de dos de las características de puntos geométricos más ampliamente utilizados es la curvatura estimada de la superficie subyacente y normal en un punto de consulta p. Ambos se consideran características locales, ya que caracterizan un punto utilizando la información proporcionada por sus k vecinos más cercanos. Para determinar estos vecinos de manera eficiente, el conjunto de datos de entrada generalmente se divide en fragmentos más pequeños utilizando técnicas de descomposición espacial como octrees o kD-trees (ver la figura a continuación - left: kD-tree, right: octree) y luego se realizan búsquedas de punto más cercanas en ese espacio dependiendo de la aplicación, puede optar por determinar un número fijo de k puntos en la vecindad de p, o todos los puntos que se encuentran dentro de una esfera de radio r centrada en p. Indiscutiblemente, uno de los métodos más sencillos para estimar las normales de superficie y los cambios de curvatura en un punto p es realizar una descomposición propia (es decir, calcular los vectores propios y los valores propios) del parche de superficie del punto k-vecindad. Por lo tanto, el vector propio correspondiente al valor propio más pequeño se aproxima a la normal de superficie n en el punto p, mientras que el cambio de curvatura de superficie se estimará a partir de los valores propios como:

$\frac{\lambda_0}{\lambda_0+\lambda_1+\lambda_2}$, donde $\lambda_0 < \lambda_1 < \lambda_2$.

2.2.4. Algoritmos búsqueda de la transformación

En la búsqueda para la transformación hemos usado varios algoritmos, los cuales he ido descartando hasta dejar el que mejor se ajustaba a las especificaciones y a los resultados optimos.

2.2.4.1. SampleConsensusPrerejective

Se trata de una clase de estimación y alineación usando una rutina RANSAC con prerrequisito.

Esta clase inserta un paso de "pre rechazo" simple pero efectivo en el ciclo de estimación de pose RANSAC estándar para evitar la verificación de hipótesis de postura que probablemente sean incorrectas. Esto se logra mediante restricciones geométricas invariantes de postura locales, así como implementadas en la clase CorrespondenceRejectorPoly.

Para alinear de forma robusta modelos parciales / ocultos, esta rutina no intenta minimizar el error, sino que intenta maximizar la tasa de inlier, por encima de un umbral que se puede especificar mediante setInlierFraction().

La cantidad de pre rechazo o "ambición" del algoritmo se puede especificar usando setSimilarityThreshold () en [0,1 [, donde un valor de 0 significa desactivado, y 1 es máximo de rechazo.

2.2.4.2. IterativeClosestPointNonLinear

Es una variante de ICP que utiliza el backend de optimización Levenberg-Marquardt.

La transformación resultante se optimiza como un cuaternión. El algoritmo tiene varios criterios de finalización:

1. El número de iteraciones ha alcanzado su máximo de iteraciones impuestas por el usuario (a través de setMaximumIterations).
2. El épsilon (diferencia) entre la transformación anterior y la transformación estimada actual es menor que un valor impuesto por el usuario (a través de setTransformationEpsilon)
3. La suma de los errores cuadrados euclidianos es menor que un umbral definido por el usuario (a través de setEuclideanFitnessEpsilon).

2.3. Forma de trabajo

La forma de trabajar para unir las nubes de puntos es la siguiente:

1. Se cargan los archivos .pcd en punteros formato PointCloud de la PCL.
2. Eliminamos los datos nan de estos archivos, ya que pueden ocasionar errores y fallos en los cálculos.
3. (Opcional) Si los datos son muy grandes tenemos la opción de eliminar los puntos que estén contenidos en el plano más grande de dicha imagen, esto suele ser el suelo. Se eliminarían muchos puntos que harán mas lenta la reconstrucción, si queremos reconstruir la escena completa los dejaremos, aunque esto añadirá más tiempo a la reconstrucción.
4. (Opcional) También podemos reducir el conjunto de puntos aplicando un VoxelGrid del tamaño que hayamos configurado, esto reducira los puntos según nuestras preferencias.
5. Lo siguiente es calcular las normales de los puntos que nos quedan, estos datos los necesitaremos para los Descriptores o features.
6. Procederemos a configurar el método que usemos:
    a. Añadiremos las nubes de puntos de entrada.
    b. Agregaremos los keypoints.
    c. Añadiremos los descriptores.
    d. Configuraremos los parámetros de la función.
    e. Aplicaremos la función con dichos datos.
7. Ahora ya tendremos la matriz que nos indica diferencia en traslación de las dos nubes de puntos, la aplicamos a la nube correspondiente.
8. Mostramos por pantalla la nube inicial y la otra nube transformada.
9. Guardamos en una matriz dicha transformación para acumularla para la siguiente iteración.
10. Volvemos al paso uno con la nube dos y una nueva nube de puntos.

Una vez tenemos los datos que vamos a usar, conocemos el procedimiento a seguir podemos seguir con el desarrollo de dicho proyecto.

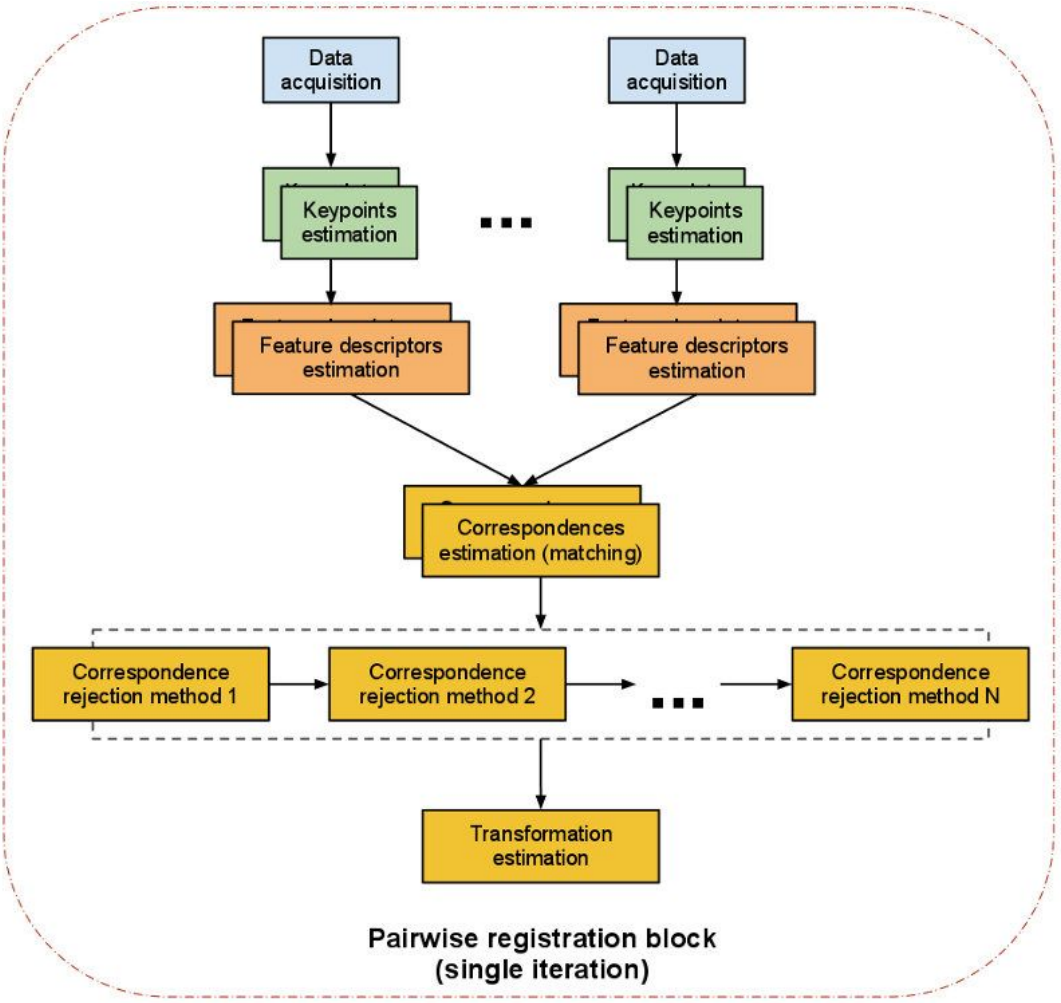


Figura 10, *Diagrama de busqueda de la transformada.*

3. Desarrollo

Comenzamos en un primer momento con la idea de hacer una interfaz con QT para un manejo sencillo. Consta de una pantalla con dos visuales, una para ver las imágenes de la Kinect y otra para la reconstrucción en tiempo real. Una vez colocada la Kinect clicamos en un botón que comenzaba la reconstrucción mientras se movía el sensor. Pero esta idea se terminó desechando ya que no se pudo configurar correctamente las herramientas necesarias para poder de ellas en una misma aplicación.

Esto fue debido a que las librerías VTK en su version 5 y 6 no se comunicaba bien con las librerias de QT, puesto que al intentar compilar junto con la PCL no encontraba algunos de los componentes necesarios, puesto que o los tenía duplicados en dos sitios diferentes o decidía no encontrarlo.

Así pues se modificaron ciertos criterios para seguir con el proyecto y se decidió poner unos requisitos.

3.1. Requisitos

3.1.1. Requisitos Funcionales:

Los requisitos funcionales nos definen las funciones del sistema de software o de sus componentes, donde cada función es un conjunto de entradas, comportamientos y salidas que define una funcionalidad específica que el software debe cumplir.

A continuación se listan los requisitos funcionales de este proyecto:

- El usuario debe de ser capaz de ver lo mismo que ve el sensor (Kinect) en la pantalla del ordenador.
- El usuario debe de poder capturar imágenes de la escena mediante un botón.
- Cada imagen que se guarde de la escena debe tener formato pcd con un nombre seguido de un número.
- La aplicación debe tener dos modos, captura de datos y reconstrucción.
- El usuario elegirá si inicia la aplicación en modo captura o en modo reconstrucción.
- La aplicación permitirá reconstruir los pcd de una carpeta en concreto.
- La aplicación debe permitir configurar los parámetros de la reconstrucción.
- La aplicación permite que se visualice un pcd.
- La aplicación permite guardar en archivo la configuración usada en la última reconstrucción.
- La aplicación dispondrá de una método para cargar y ver los datos una vez reconstruidos.
- La aplicación dispondrá de varios métodos para reconstruir parte o la totalidad de las imágenes.
- La aplicación debe de poder volver a los valores por defecto para iniciar distintas reconstrucciones.

3.1.2. Requisitos no Funcionales:

Los requisitos no funcionales son aquellos que no afectan a la funcionalidad de la aplicación pero que son exigidos por las especificaciones. En el caso de este proyecto, los requisitos no funcionales son los siguientes:

- La aplicación debe ejecutarse en sistema linux.
- El idioma de la aplicación será en Inglés para que tenga una mayor entendimiento por la mayoría de programadores.
- La aplicación deberá correr en un portatil con mínimo:
  - Asus X52JB con características:
  - Procesador Intel® Core™ i5
  - 2 x SO-DIMM socket 4 Gb SDRAM , DDR3 1066 MHz SDRAM
  - ATI Mobility™ Radeon® HD5145 512MB DDR3
  - Puertos usb 2.0
- La aplicación deberá comunicarse con una Kinect v1.0.
- Se necesita de un enchufe para la Kinect.

Software:

- c++
- PCL 1.7 mínimo

Para configurar el pc con las herramientas básicas he seguido este tutorial.

Configuración del ordenador.

Para instalar c++ usamos:

```
sudo apt-get install build-essential
```

Lo siguiente seria el compilador cmake:

```
sudo apt-get install cmake
```

Para los drivers de la Kinect necesitaremos el package *freenect*, que se encuentra en el apt-get, la instalaremos con el siguiente comando en linux:

```
sudo apt-get install freenect
```

Por último instalaremos la PCL 1.7 usando estos comandos en linux:

```
sudo apt-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
sudo apt-get update
sudo apt-get install libpcl-all
```

Para la reconstrucción a tiempo real necesitamos de un ordenador con gráfica Nvidia para el procesamiento GPU, ya que el portátil con el que se realiza este proyecto está limitado y no dispone de dicha gráfica, se descarta esta idea. Es entonces cuando decidimos usar como herramientas para tratar las nubes de puntos la PCL y sus métodos, esto será más lento está dentro de nuestro planning.

Ahora que ya tenemos el ordenador configurado y sabemos de los requerimientos tanto funcionales y no funcionales podemos centrarnos en la implementación del mismo.

A continuación buscamos proyectos, ejemplos o tutoriales sobre la reconstrucción con PCL y lo que hay hasta el momento necesita de ordenadores potentes, de una herramientas de visión con gran precisión y escenarios muy concretos o solo funcionan con los datos proporcionados, una vez cambias los datos de las entradas dichos programas no se ajustan bien y toca modificar su código y volver a compilarlo mediante ensayo y error.

Un ejemplo que se encontró fue "*Robust pose estimation of rigid objects*[1] ", éste tutorial se usa para encontrar y alinear el modelo de un muñeco chef en una escena, todo esto en una escena muy pequeña con pocos datos que puedan interferir en la búsqueda de la alineación del modelo. Se usa Uniform sampling para los keypoints, NormalPoint como normales y NormalEstimationOMP como metodo para su cálculo, Descriptores FPFHSignature33 y SampleConsensusPrerejective para alinear las nubes de puntos.

El primer prototipo de la aplicación tenía estos mismos elementos a excepción de los descriptores, los cuales se cambiaron por SHOT32, que funcionan mejor con escenarios grandes.

Tras varias pruebas con datos de entrada diferentes y distintas configuraciones se encontró una configuración que funcionaba bien, pero tenia un elemento de aleatoriedad que no hacía que fuese fiable y que a veces fallase; en la parte de resultados veremos dicha reconstrucción y los parámetros que se usaron.
Encontramos varios problemas ya que no fallaba en la primera reconstrucción pero si en las siguientes y fueron apareciendo más problemas al guardar las Point Clouds, puesto que al recuperar los datos para su posterior visualización, no se

guardaban las transformaciones con respecto a la PointCloud inicial y no se colocaban bien en el espacio tridimensional.

Otro problema que apareció fue que según las características de los datos había que variar la configuración conforme aumentaban las iteraciones, es decir si habíamos unido las primeras tres Point Clouds al querer unir la cuarta fallaba. Modificando las cantidades de keypoints, descriptores y demás configuraciones se solucionaba dicho error, pero hay que estar muy atentos al proceso y puesto que tarda bastante tiempo en buscar la transformación no nos servia, nosotros queremos que haga una reconstrucción aunque no sea perfecta, pero sin variar los parámetros una vez configurados.

Se tuvo que descartar esta metodología, ya que no se conseguía unos buenos resultados y también necesitaba demasiado tiempo (tarde entera). Otro ejemplo fue el tutorial sobre cómo registrar incrementalmente pares de nubes de puntos. En el tutorial se usan 5 Point Clouds de una pared con escaleras, una silla y un escritorio, éste código junta Point Clouds dos a dos y las va guardando en un archivo, que posteriormente tenemos que visualizarlo con la herramienta "*pcl_viewer*" la cual le da un color a cada PointCloud de la siguiente forma:

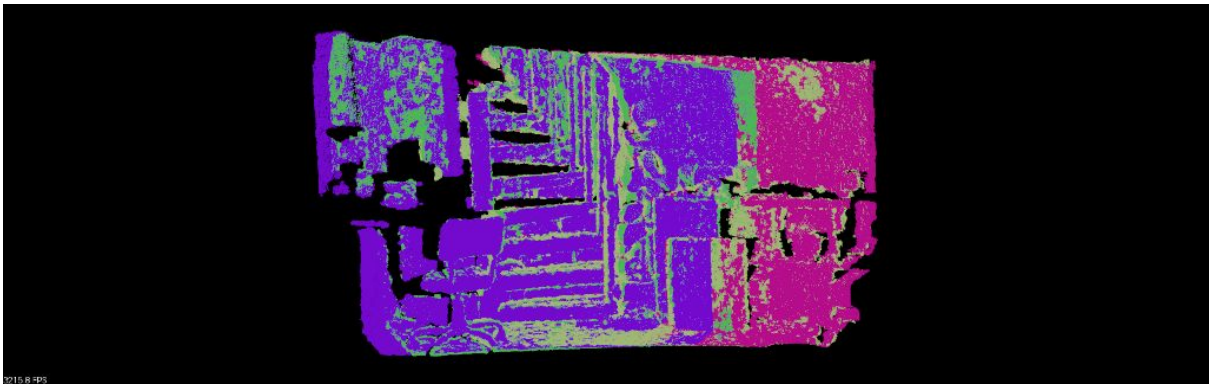


Figura 11, *Unión de las 4 imágenes del Tutorial [4].*

Es entonces cuando usamos decidimos usar "*IterativeClosestPointNonLinear"* ya que los demás métodos no llegaban a funcionar correctamente. Una vez implementado el nuevo código se comenzó a buscar la mejor reconstrucción posible. Tras varias configuraciones y pruebas encontramos más rápidamente configuraciones que hacían que la reconstrucción fuera bastante acertada.

Para éste método se usa como keypoints la totalidad de los datos, aunque se ha añadido una opción para hacer un voxelGrid y reducir la cantidad de información. Las normales se siguen calculando, pero en cuanto a los descriptores no se usan como tal, ya que para el método se necesitan como datos un puntero de tipo PointCloud que contiene los puntos normales y los datos de las normales, con estos datos le es suficiente para encontrar la matriz.

Dicho método necesita de:

- ***setTransformationEpsilon***: Máxima distancia de ajuste entre dos transformaciones consecutivas, para considerar que una optimización ha convergido a la solución final.
- ***setMaxCorrespondenceDistance***:Este parámetro es la máxima distancia entre dos correspondencias.
- ***setMaximumIterations***: Máximas Iteraciones de ICP.

Seguidamente realizaremos unas iteraciones con el fín de encontrar una posible solución. Iremos mejorando esta solución parcial hasta encontrar la óptima. Durante cada iteración, hacemos un seguimiento y acumulamos las transformaciones devueltas por el ICP.

Si la diferencia entre la transformación encontrada en la iteración N y la encontrada en la iteración N-1 es menor que el umbral de transformación pasado a ICP, refinamos el proceso de coincidencia eligiendo correspondencias más cercanas entre el origen y el objetivo.

Una vez que se ha encontrado la mejor transformación, la invertimos (para obtener la transformación del objetivo a la fuente) y la aplicamos a la nube objetivo.

El objetivo transformado luego se agrega a la fuente y se devuelve a la función principal con la transformación.

La evolución del cuerpo principal ha sido la anteriormente mencionada, pero también he tenido que superar otras dificultades, como por ejemplo que al guardar los datos si no se hacía de forma binaria en pcd éste no mantenía la traslación que se le aplicaba, por lo que al intentar recuperar los datos resueltos no se colocaban correctamente, sin embargo estaban bien en la reconstrucción antes de ser guardados. Otro problema estaba ligado a la cantidad de datos que podia tratar el visualizador, teniendo que aplicar una reducción de datos para poder mostrarlos, pero a la vez se mantenían los datos para que fuesen guardados en su mejor calidad y cantidad. Estos datos si se intentaban guardar en un único archivo se perdía muchísima información y no se podia distinguir si la reconstrucción era buena o no, el resultado era muy difuso, es por ello que se guarda en varios archivos, concretamente un archivo por cada unión de dos nubes de puntos pcd.

La aplicación se ha hecho para que en su menú pudiesemos configurar todos los parámetros de la reconstrucción y no tener que estar compilando por cada cambio en dichos métodos, ya que sabemos que según los datos que intentemos reconstruir necesitaremos muchos cambios en las configuraciones. En los tutoriales y ejemplos esto no se hace, por lo que demora mucho más tiempo si queremos probar con nuestros propios datos.

La aplicación comienza mostrandonos estos datos, que son los referentes a los parámetros de inicialización del mismo.

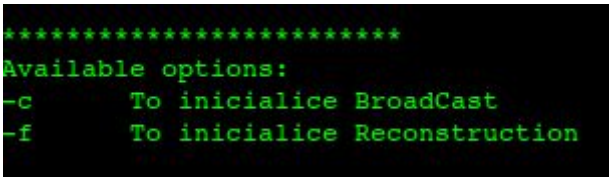


Figura 12, *Parámetros de inicialización de la aplicación.*

Ahora nos muestra las posibles opciones que nos da la aplicación:

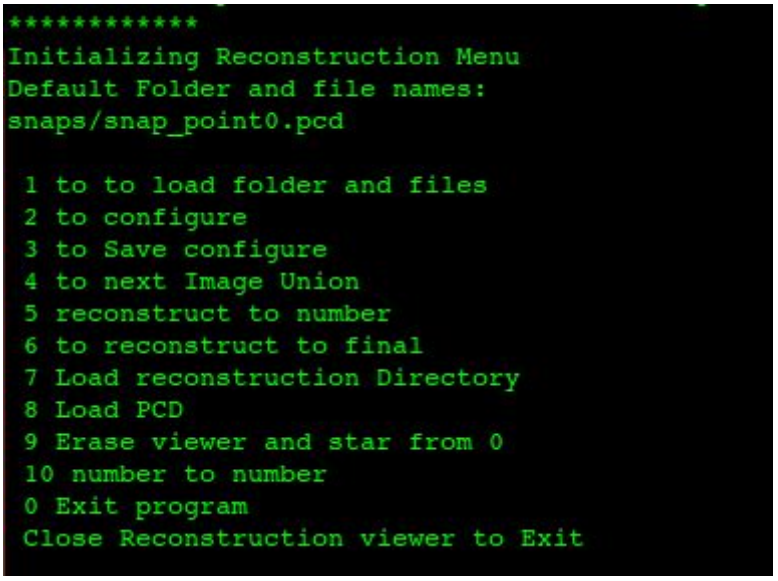


Figura 13, *Menú Básico de la aplicación.*

Todas estas opciones han sido creadas para que no tengamos que compilar, cerrar y volver a iniciar visualizadores y que podamos trabajar con los datos mas cómodamente.

En la siguiente figura se muestra el menú de configuración en el que se muestran los parámetros por defecto y se nos deja configurarlos a nuestro antojo. Aquí están todos aquellos parámetros que influyen en una reconstrucción.

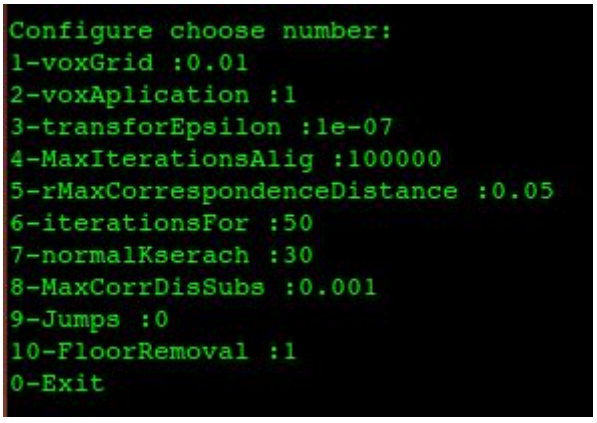


Figura 14, *Menú configuración.*

4. Resultados

En este apartado se va mostrar los resultados de las pruebas de los dos métodos usados para la reconstrucción.

4.1. Primer prototipo.

El primer prototipo de la aplicación usaba el método “SampleConsensusPrerejective”, y necesitaba de los siguientes parámetros de configuración:

- **model_ss_**: Éste parametro se encarga de escoger los keypoints de la Point Cloud perteneciente a la imágen que vamos a transformar. Usa UniformSampling.
- **scene_ss_** : Escoge los keypoints de la imágen que usamos como escenario.
- **descr_rad_**: Se trata del radio de la esfera para calcular los descriptores “features”.
- **ransalterations**: Número de iteraciones de RANSAC.
- **rNumberOfSamples**: Establece el número de muestras de ejemplo para usar durante las iteraciones.
- **rCorresRandom**: Establezca la cantidad de vecinos a usar cuando seleccione una correspondencia de características aleatorias.
- **rSimilarityThreshold**: Establece el umbral de similitud entre los bordes del objeto poligonal.
- **rMaxCorrespondenceDistance**: Umbral de entrada
- **rInlierFraction**: Fracción de inliers requerida para aceptar una hipótesis de postura.

Los primeros intentos de reconstrucción se guardaban todas las iteraciones en un único tipo Point Cloud, esto hacía que fuese muy pesado y que en varias iteraciones fallara la aplicación.

Aquí tenemos una iteración con dicho código usando la mayoría de los datos:

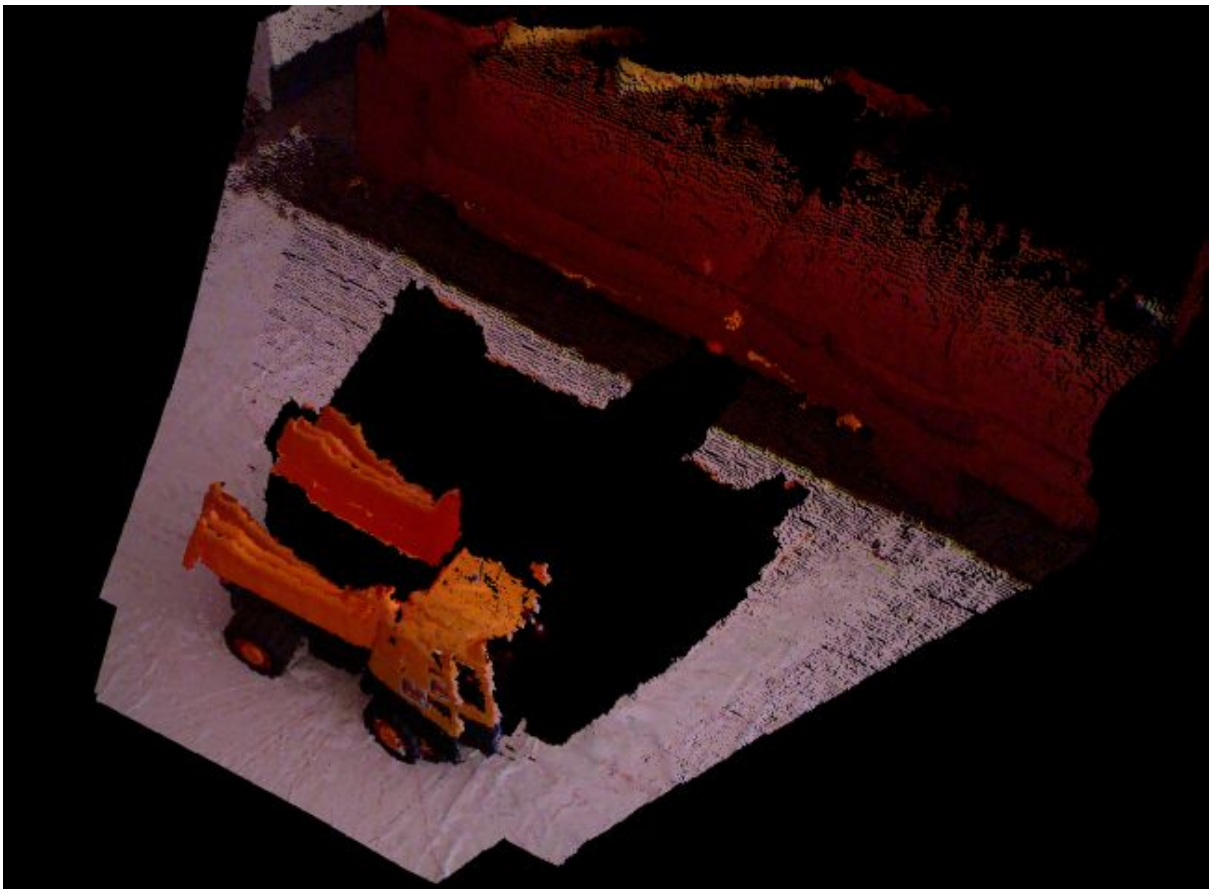


Figura 15, *Camión dos iteraciones.*

Configuración de la reconstrucción:

- model_ss_ :0.01
- scene_ss_ :0.01
- descr_rad_ :0.1
- ransaIterations :50000
- rNumberOfSamples :3
- rCorresRandom :5
- rSimilarityThreshold :0.9
- rMaxCorrespondenceDistance :0.0125
- rInlierFraction :0.25

Como se ve en la imágen no es muy exacta la unión, ya que está girado y se ven dos cabezas del camión.

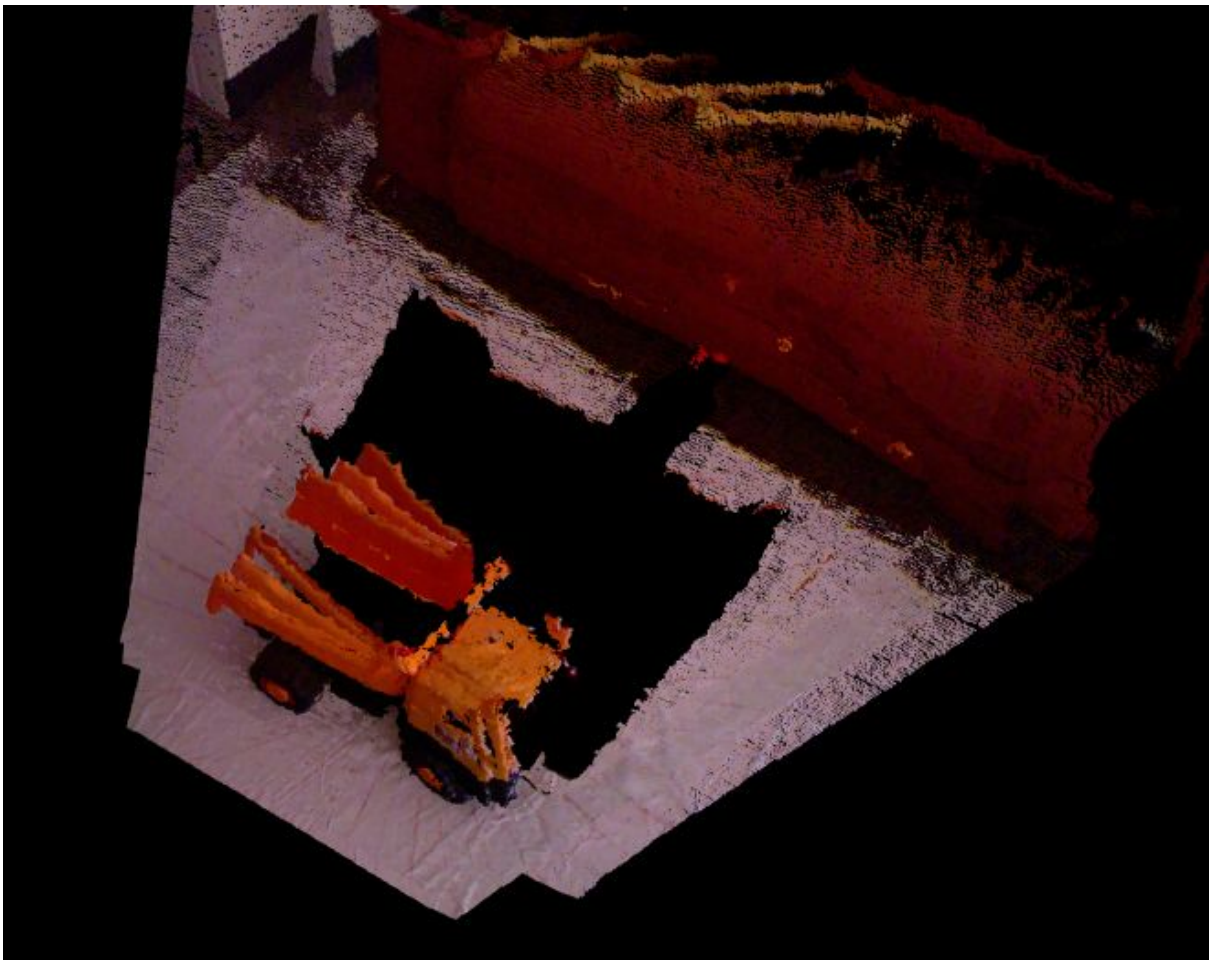


Figura 16, *Camión tres iteraciones.*

- model_ss_ :0.01
- scene_ss_ :0.02
- descr_rad_ :0.08
- ransaIterations :50000
- rNumberOfSamples :3
- rCorresRandom :5
- rSimilarityThreshold :0.9
- rMaxCorrespondenceDistance :0.0125
- rInlierFraction :0.25

Si seguimos con la reconstrucción vemos en la figura 16 como el camión tiene tres o incluso cuatro paredes en la parte trasera. Esta configuración no sirve para seguir iterando como vemos si avanzamos en su reconstrucción en la figura 17.

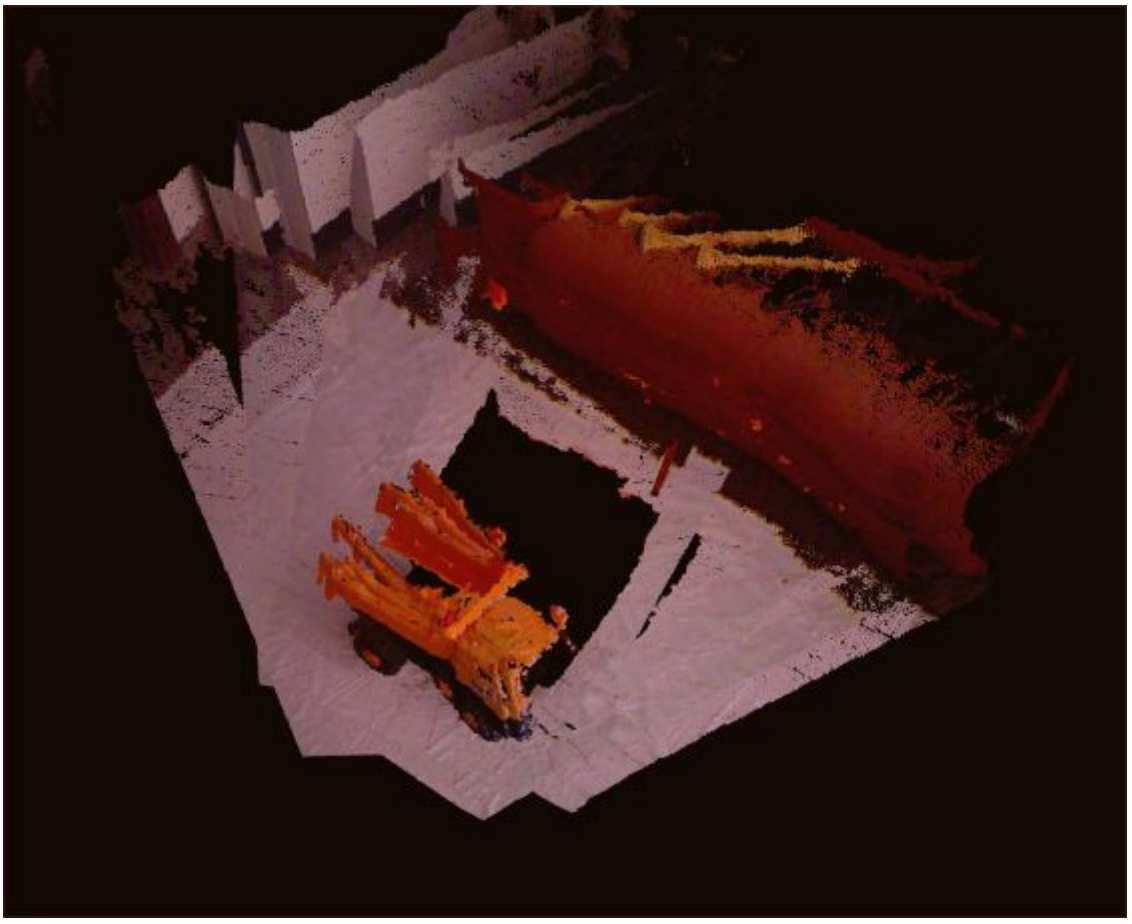


Figura 17, *Camión seis iteraciones.*

Tras varios cambios relajamos la distancia de correspondencia de 0.0125 a 0.9, haciendo que se ajustara mejor, ya que intentando variar los otros parámetros no se conseguía nada mejor que lo ya encontrado. En las figuras 18 y 19 vemos una iteración y tres iteraciones con dicha configuración.

Figura 18, *Camión una iteración.*

Figura 19, *Camión tres iteraciones.*

En comparación con las primeras pruebas se nota que funciona mucho mejor pero al llegar a tres iteraciones (figura 19 vemos que empiezan a aparecer datos erróneos en mitad del aire, hasta aquí no está nada mal pero como he dicho antes en el momento que vamos aumentando las iteraciones hay demasiados puntos, por eso en la siguiente reconstrucción hemos eliminado los puntos pertenecientes el plano más grande, que en éste caso es el suelo. También hemos cambiado un poco los parámetros debido a que han cambiado la cantidad de puntos:

- model_ss_ :0.01
- scene_ss_ :0.02
- descr_rad_ :0.6
- ransaIterations :500000
- rNumberOfSamples :3
- rCorresRandom :5
- rSimilarityThreshold :0.9
- rMaxCorrespondenceDistance :0.03
- rInlierFraction :0.25

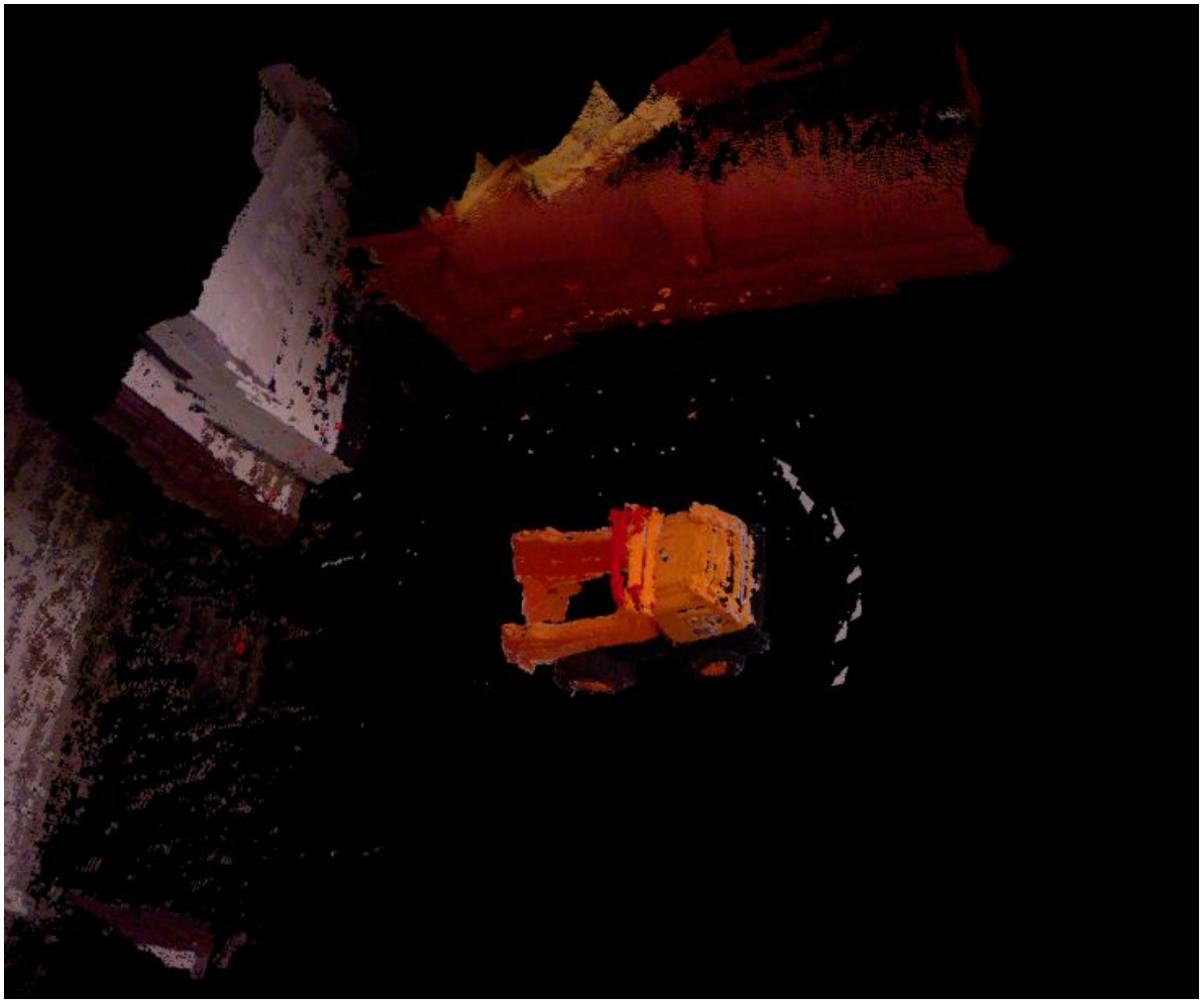


Figura 20, *Camión quince iteraciones.*

Al llegar a quince iteraciones vemos la acumulación de puntos en la esquina de la cabina del camión debido al error en la reconstrucción. Ahora podemos intentar reconstruir los datos de la persona.

- model_ss_ :0.01
- scene_ss_ :0.02
- descr_rad_ :0.08
- ransaIterations :50000
- rNumberOfSamples :3
- rCorresRandom :5
- rSimilarityThreshold :0.9
- rMaxCorrespondenceDistance :0.09
- rInlierFraction :0.25

Figura 21, *Persona una iteración.*

Figura 22, *Persona tres iteraciones.*

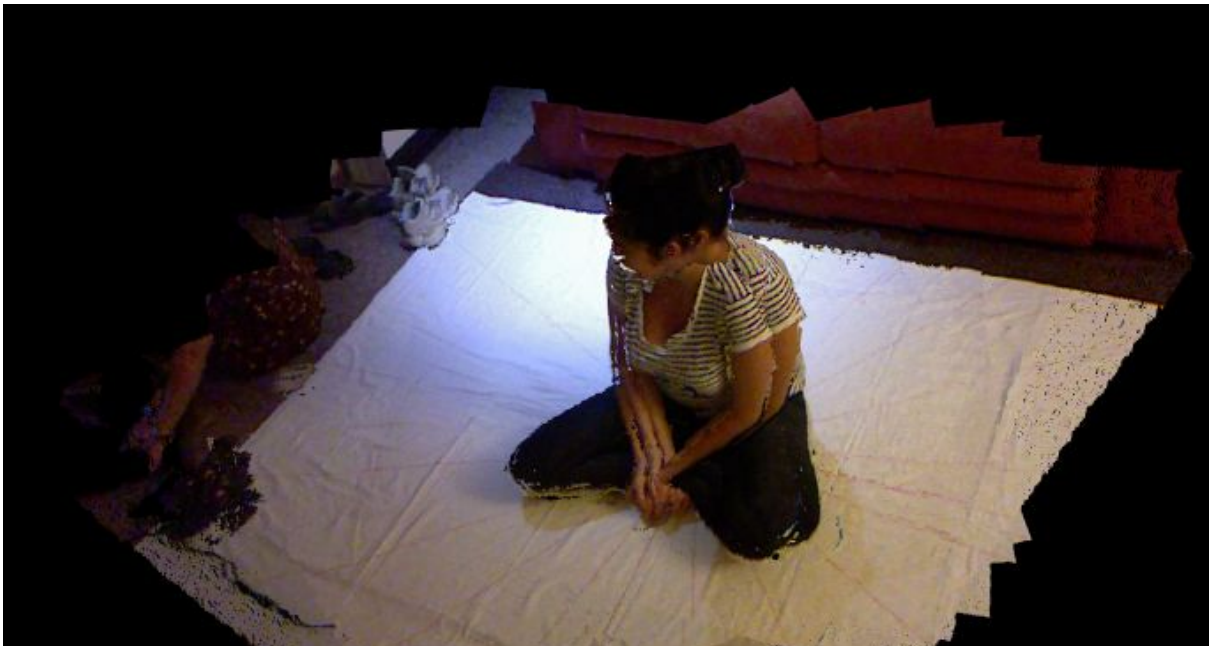


Figura 23, *Persona seis iteraciones.*

Aquí ya vemos que se hace patente esa aleatoriedad que he mencionado anteriormente. si eliminamos el suelo podemos llegar a once iteraciones, aunque variando la configuración ya que hay menos puntos.

- model_ss_ :0.03
- scene_ss_ :0.04
- descr_rad_ :0.6
- ransaIterations :500000

- rNumberOfSamples :3
- rCorresRandom :5
- rSimilarityThreshold :0.9
- rMaxCorrespondenceDistance :0.015
- rInlierFraction :0.25

Figura 24, *Persona seis iteraciones.*

Pero al seguir aumentando ya se vuelve a ver la aleatoriedad en la que aparecen 4 piernas, giros que colocan el suelo con inclinación y peores resultados.

Como vemos en la figura 24b marcado, la reconstrucción comete un error y coloca una de las nubes mal, por lo que el modelo aparece con dos cabezas y tres piernas.

Figura 24b, *Persona hasta final de iteración, fallo 2 cabezas.*

Ahora se mostrarán otros errores.

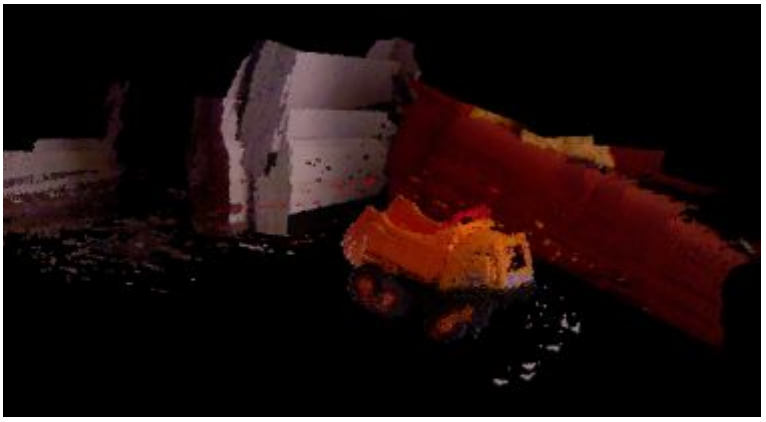


Figura 25, *Camión y error de ángulo.*

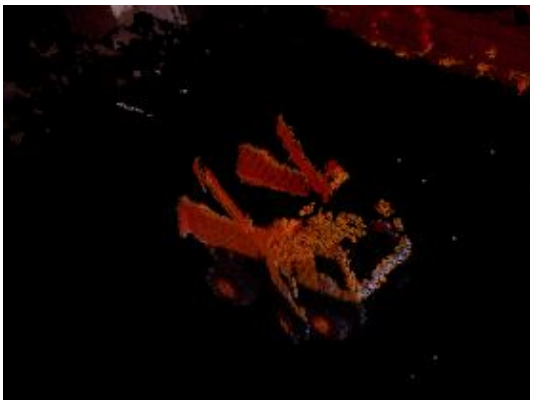


Figura 26, *Camión y error de ángulo.*

4.2. Aplicación final

La aplicación final usa el método “IterativeClosestPointNonLinear”, y necesitaba de los siguientes parámetros de configuración:

- **voxGrid**: este parámetro se encarga de rebajar la cantidad de keypoints.
- **voxAplication**: este parámetro nos dice si usamos el voxGrid o usamos todo el conjunto de datos.
- **transforEpsilon**: Máxima distancia de ajuste entre dos transformaciones consecutivas, para considerar que una optimización ha convergido a la solución final.
- **MaxIterationsAlig**: Máximas Iteraciones de ICP.
- **rMaxCorrespondenceDistance**: Este parámetro es la máxima distancia entre dos correspondencias.
- **iterationsFor**: Número de iteraciones para intentar mejorar a mano el resultado obtenido.
- **normalKserach**: Configuración para el cálculo de las normales.
- **MaxCorrDisSubs**: parametro que modifica la distancia de las correspondencia en la búsqueda a mano.
- **FloorRemoval**: Nos indica si queremos remover los puntos pertenecientes al plano de mayor tamaño en los datos, en nuestros datos es el suelo (si apuntamos con la cámara a una pared, ésta sería el plano de mayor tamaño).

Comenzamos intentando unir los datos del camión ya que estos tienen menos errores y están tomados con mayor exactitud.

Tras varios intentos se logra encontrar una configuración muy rápidamente comparada con el prototipo inicial, que funciona para una reconstrucción total.

Figura 27a, *Camion reconstruido totalmente.*

- voxGrid :0.01
- voxAplication :0
- transforEpsilon :1e-10
- MaxIterationsAlig :1000
- rMaxCorrespondenceDistance :0.03
- iterationsFor :20
- normalKserach :30
- MaxCorrDisSubs :0.002
- FloorRemoval :1

Estos parámetros nos indican que usamos la totalidad de los puntos como keypoints, también que hacemos pocas iteraciones con ICP para encontrar un resultado y que nos sirve uno que sea mas o menos decente, y que su correspondencia sea mejor. Para las iteraciones manuales incrementamos la distancia porque hemos cogido un resultado no tan perfeccionista y a cada iteración lo vamos mejorando.

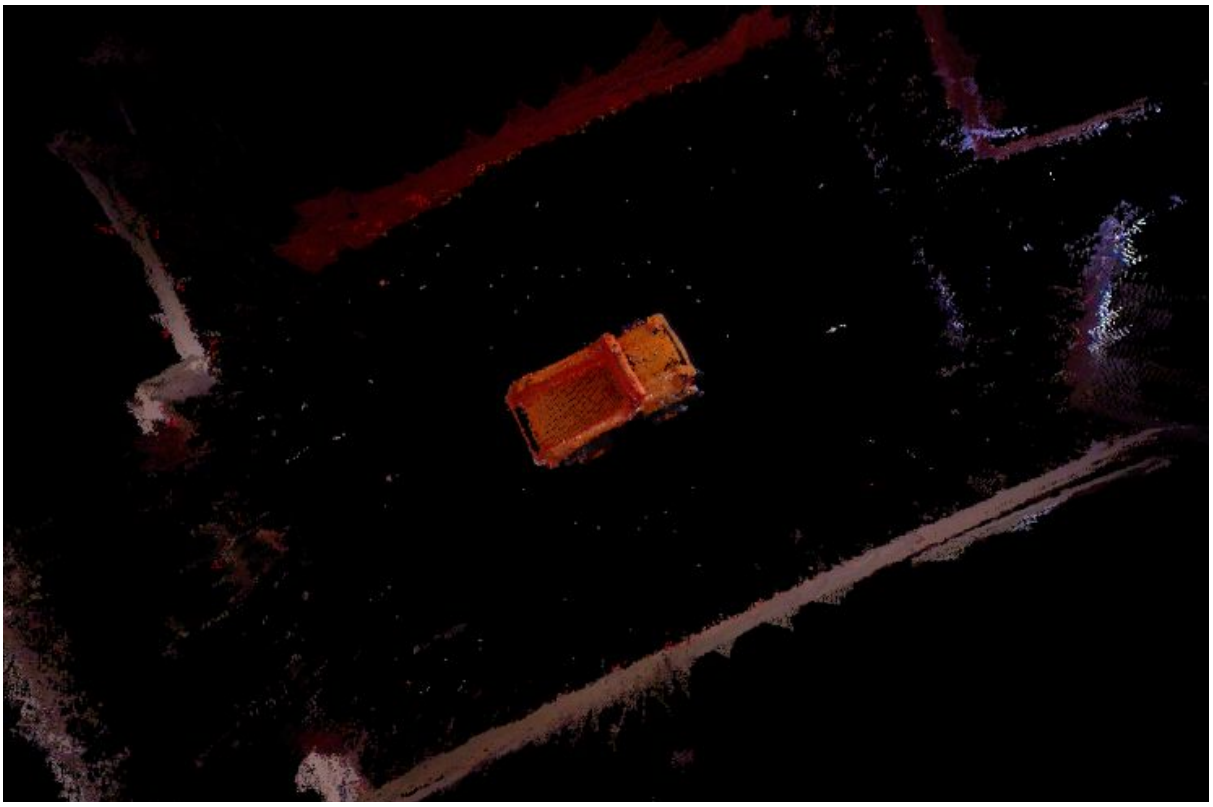


Figura 27b, *Camion reconstruido totalmente.*

Figura 27c, *Camion reconstruido totalmente.*

Ahora intentaremos reconstruir los datos de la persona, hay que tener en cuenta que estos datos están tomados de forma menos precisa y de forma más erratica, como lo tomaria una persona dándole unas instrucciones poco precisas de como tomar dichos datos.

- voxGrid :0.01
- voxAplication :1
- transforEpsilon :1e-07
- MaxIterationsAlig :100000
- rMaxCorrespondenceDistance :0.05
- iterationsFor :50
- normalKserach :30
- MaxCorrDisSubs :0.001
- FloorRemoval :1

Con estos parámetros logramos reconstruir el total de los datos.

Figura 28, *Persona Reconstrucción.*

5. Conclusiones

Con los ejemplos y tutoriales actuales para el tratamiento de datos pcd se ha logrado reunir información y hemos llegado a obtener un resultado de aplicación que logra el objetivo previsto, reconstruir datos en 3D. En cuanto a la precisión de la reconstrucción depende mucho de la calidad de los datos y de la forma en que se toman los mismos.

Se ha podido terminar una aplicación con diferentes funciones que la hacen diferente a lo que hay hasta el momento, permitiendo configurar, probar y visualizar parte de las reconstrucciones haciendo que disminuya el tiempo para la reconstrucción total de unos datos y consiguiendo calidades de las mismas según nuestras necesidades.

Estoy satisfecho con el resultado ya que he conseguido reconstruir varios datos de distinto tamaño, cosa que parecía muy improbable al comienzo del proyecto. Es una aplicación muy sencilla a la cual se le pueden ir añadiendo mejoras que la hagan mucho más funcional y completa. Han habido varias ideas que no he podido implementar debido al tiempo y a los problemas encontrados. Se conoce del error al terminar la reconstrucción en el que no coloca la última foto en el lugar adecuado debido a la acumulación de errores en el ángulo. Esto se podría mejorar calculando la distancia entre el primer y el último conjunto de datos, si logramos ver esta diferencia la dividimos entre la cantidad de imágenes y los vamos sumando, haciendo que se coloquen los datos y se ajustaran mejor.

Otra mejora sería disponer de una GPU y distribuir los cálculos en la gráfica, haciendo que reconstrucciones que tardan unas dos o tres horas tarden minutos. Aunque habría que comprobar si realmente sale rentable o no.

En cuanto a la forma de guardar los datos, estos se podrían mejorar, haciendo un formato en el que guardará en el mismo archivo tanto la nube de puntos como la matriz de posicionamiento, haciendo posible una regresión y modificación de la matriz de posicionamiento para arreglar el error final.

Si disponemos de un sensor que tenga menores errores a la hora de tomar los datos y que no necesitase un espacio concreto para empezar a poder tomar datos esto haría que se pudiese reconstruir otro tipo de datos, o incluso reconstruir superficies pequeñas pero con mucha precisión.

En resumen y para concluir, este trabajo ha sido una grata experiencia de la cual espero seguir modificando para sacarle mejor partido en un futuro y quien sabe si se podria usar en algún lugar o si me logro hacer con una impresora 3D podría hacer lo mísmo que los personajes de la serie Big Bang Theory e imprimir un muñeco de mi mismo modelado con mi aplicación.

5.1. Figuras Adicionales

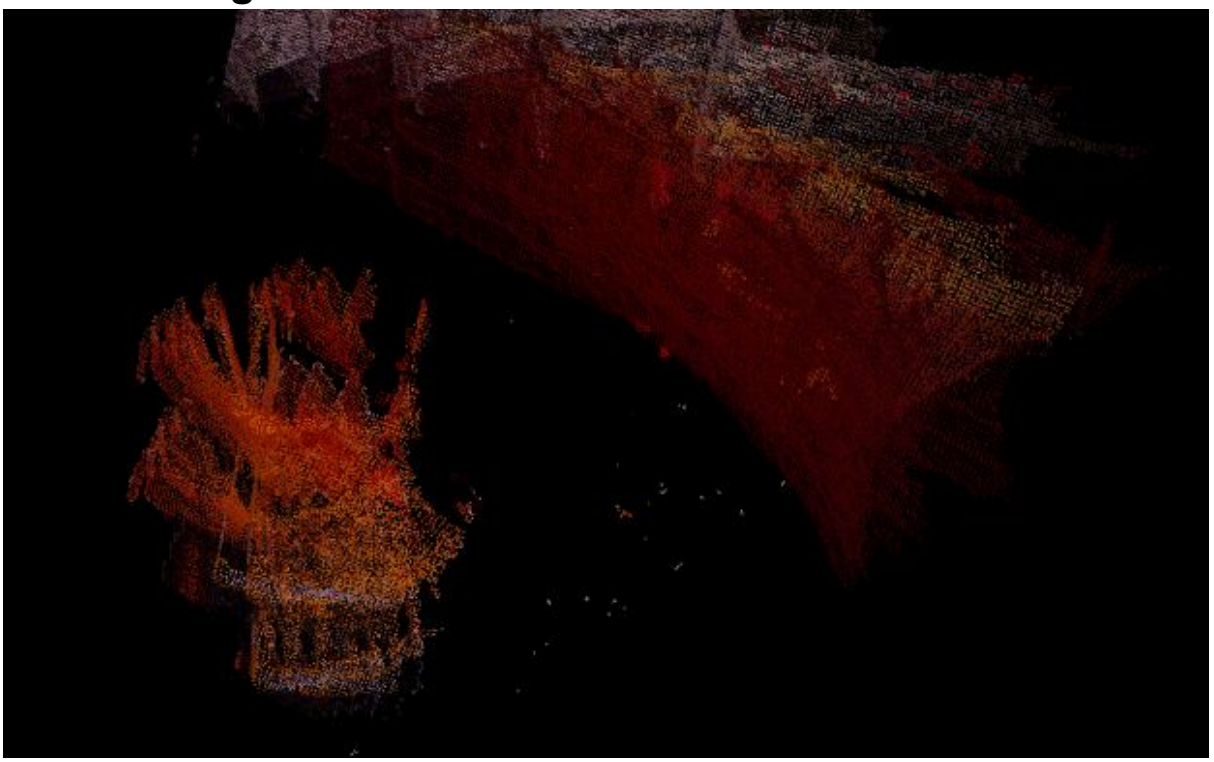


Figura 29, *Guardado no binario.*

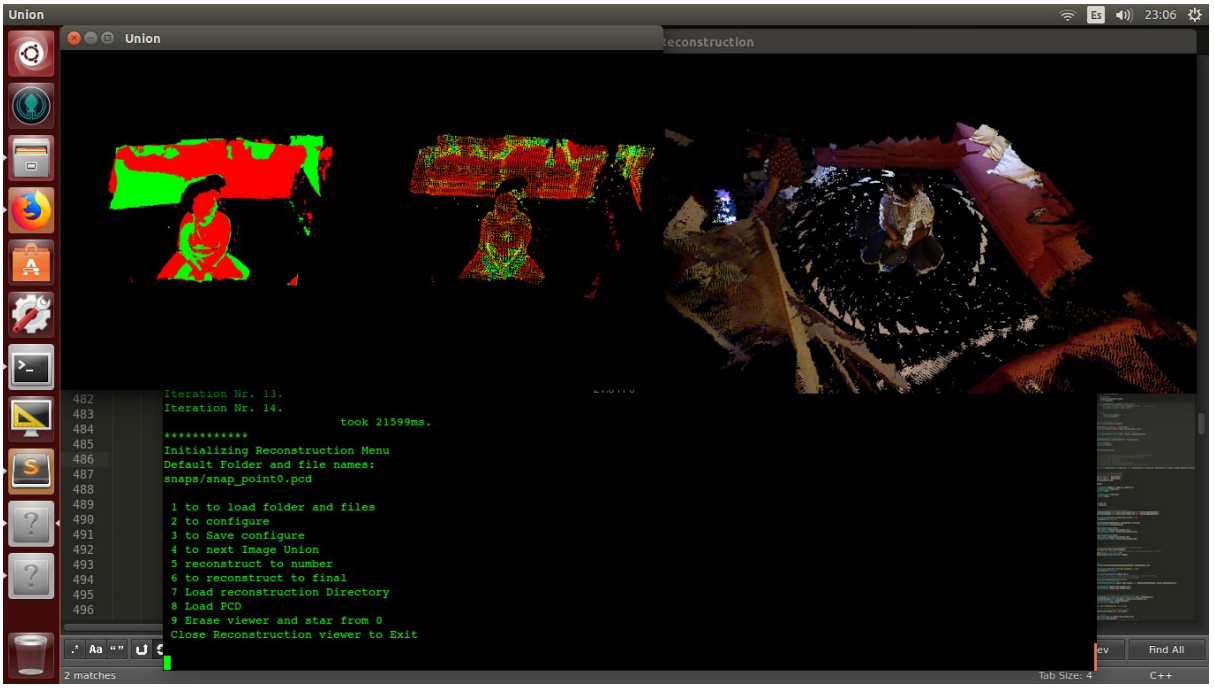


Figura 30, *Formato Aplicación Final.*

6. Bibliografía

[1] PCL Documentation. ***Robust pose estimation of rigid objects.***
10 Septiembre 2017, en
http://pointclouds.org/documentation/tutorials/alignment_prerejective.php

[2] PCL Documentation. ***What is a Point Cloud?.***
1 de Diciembre 2017, en http://pointclouds.org/about/.

[3]PCL Documentation. ***Group keypoints***.
1 de Diciembre 2017, en http://docs.pointclouds.org

[4]PCL Tutoriales. ***How to incrementally register pairs of clouds***
10 de Diciembre 2017 , en
http://pointclouds.org/documentation/tutorials/pairwise_incremental_registration.php